Performance Evaluation of MPI on Cray XC40 Xeon Phi Systems

Scott Parker
Argonne National Laboratory
sparker@anl.gov

Sudheer Chunduri
Argonne National Laboratory
sudheer@anl.gov

Kevin Harms
Argonne National Laboratory
harms@anl.gov

Krishna Kandalla
Cray Inc.
kkandalla@cray.com

ABSTRACT

The scale and complexity of large-scale systems continues to increase, therefore optimal performance of commonly used communication primitives such as MPI point-to-point and collective operations is essential to the scalability of parallel applications. This work presents an analysis of the performance of the Cray MPI point-to-point and collectives operations on the Argonne Theta Cray XC Xeon Phi system. The performance of key MPI routines is benchmarked using the OSU benchmarks and from the collected data analytical models are fit in order to quantify the performance and scaling of the point-to-point and collective implementations. In addition, the impact of congestion on the repeatability and relative performance consistency of MPI collectives is discussed.

INTRODUCTION

Given the technological trends in the performance of the compute and network components of HPC systems [6], the performance and scaling of parallel applications on these systems is highly dependent on their communication performance. Optimal implementation and usage of MPI point-to-point and collective routines is essential for the performance of MPI based applications. In this study the performance of MPI point-to-point and collectives routines on the Theta supercomputer at the Argonne Leadership Computing Facility (ALCF) is evaluated using the OSU benchmarks. Theta is a Cray XC40 system equipped with the Cray Aries interconnect in a Dragonfly topology and a proprietary Cray MPI implementation derived from the open source MPICH implementation . The compute nodes on Theta utilize the Intel Xeon Phi Knights Landing processor.

Numerous models for MPI point-to-point performance exist and in this work the Hockney [2] or "postal" model is utilized to represent the latency of MPI point-to-point operations. In an MPI library the collective routines are implemented using a sequence of point-to-point message exchanges. Different message patterns or algorithms are used for the different collectives, and within the same MPI collective different patterns are used for different messages sizes and node counts. Well established models for collectives performance have been defined [7] and are used as a basis for fitting analytic models to the measured collective latency data. Additionally, the impact of network congestion on MPI performance is quantified using an MPI benchmark that has been run repeatedly on Theta on different days under different load conditions. Finally, MPI performance consistency guidelines that have been defined in the literature [3, 8, 9] were tested on Theta and the adherence of the Cray MPI implementation on Theta to these guideline was evaluated.

THETA SYSTEM DESCRIPTION

The ALCF Cray XC40 system Theta is an 11.7 peta-flop system that utilizes the second generation Intel Xeon Phi Knights Landing many core processor and the Cray Aries interconnect. A high level system description is given in Table 1. The system consists 4,392 compute nodes with an aggregate 281,088 cores with the nodes housed in 24 racks. The compute nodes are connected using a 3-tier dragonfly topology with two racks creating a dragonfly group, resulting in a total of twelve groups.

Aries Network

The Cray XC series utilizes the Cray Aries interconnect, a follow on to the previous generation Gemini interconnect. Aries utilizes a system-on-chip design that combines four network interface controllers (NIC) and a 48 port router onto a single device which connects to four XC compute nodes via a 16x PCI-Express Gen3 connection. Each NIC is connected to two injection ports on the router and 40 ports are available for links between routers. The PCIe interface provides for 8 GT/s per direction with 16 bits for a total of 16 GB/s of peak bandwidth. The network links may be electrical or optical with the electrical links providing 5.25 GB/s of bandwidth per direction and the optical links providing 4.7 GB/s.

A node initiates network operations by writing across the host interface to the NIC. The NIC then creates packets containing the request information and issues them to the network with packets containing up to 64 bytes of data. Aries implements two messaging protocols: fast memory access (FMA) and block transfer engine (BTE). FMA offers minimized overhead for 8-64 byte operations resulting in a fast path for single word put, get and non-fetching atomic operations but requires the CPU be involved in the message transfer. This provides low latency and a fast issue rate for small transfers. Writes to the FMA window produce a stream of put operations each transferring 64 bytes of data. The BTE is used for larger messages and can result in higher achieved bandwidth and provides for asynchronous transfers independent of the CPU, however higher messages latencies exist. To utilized the BTE a process writes a block transfer descriptor to a queue and the Aries hardware performs the operation asynchronously. Up to four concurrent block transfers are supported allowing maximum bandwidth to be achieved for smaller concurrent transfers. Block transfers have higher latency than FMA transfers but can transfer up to 4GB without CPU involvement. Aries supports atomic operations, including put operations such as atomic add, and get operations such as conditional swaps, and maintains a 64 entry atomic operation cache to reduce host reads when multiple processes access the same variable. These network atomics are not coherent with respect to local memory operations. Additionally, the NIC contains a collective engine that provides hardware support for reduction and barrier

Processor core	KNL (64-bit)
CPUs per node	1
# of cores per CPU	64
Max nodes/rack	192
Racks	24
Nodes	4,392
Interconnect	Cray Aries Dragonfly
Dragonfly Groups	12
# of links between groups	12

Table 1: Theta – Cray XC40 system

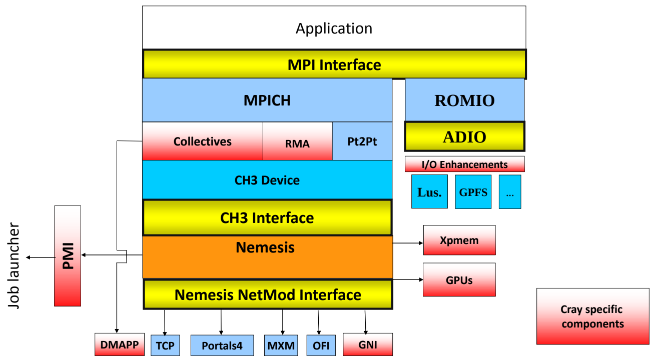

Figure 1: Cray MPI

operations which are optimized for latency sensitive single word operations and supports logical, integer, 32 and 64 bit floating point operations.

The Aries router is organized into 48 tiles in a a 6x8 matrix. Eight tiles, referred to as processor tiles, are associated with the four NICs with four processor tiles shared between NIC0 and NIC1 and four shared between NIC2 and NIC3. The other forty tiles are network tiles which consist of an input queue, a sub-switch, and a column buffer. The input queue receives packets and sends them across a row bus to the target sub-switch in a column. The sub-switches receive packets from the eight input queues in a row and switches packets to the appropriate output sending them on the column bus to the column buffer in the appropriate output row. The column buffer collects packets from the six tiles in each column and multiplexes them on the output. The Aries router contains four routing tables, one each for local minimal, local non-minimal, global minimal and global non-minimal routes and randomly generates two minimal and non-minimal routes for every packet. Path load information is used to select the least loaded path.

The XC series network is configured in Dragonfly topology. A dragonfly network consists of multiple groups, with Theta having 12 groups. Within a group, which consists of 96 Aries and up to 384 compute nodes, Aries routers are connected in a 6x16 two-dimensional grid. In this arrangement each Aries router is connected to each of the other Aries in the same row and column using a single link between each router. This provides all-to-all connectivity between the Aries routers in the same rows and columns. The links within a group are electrical. In addition to the intra-group links between rows and columns routers may provide optical inter-group links. Multiple links may be used to connect any two groups and Theta has 12 optical links between each group. This topology has a maximum of five hops between nodes, and minimum of zero hops for minimal routes. A zero hop route does not traverse any network links and stays within the nodes connected directly to the same Aries router. A five hop route between groups performs two hops within a group to reach a router containing a link to the other group and two hops with the destination group to reach the destination node. Traffic may be routed along a minimal route between nodes, or in the face of congestion along a non-minimal route where packets are routed through a randomly selected intermediate router.

Cray MPI

Cray MPI is a proprietary implementation of the Message Passing Interface (MPI) specification, optimized for the Cray Aries interconnect. Cray MPI is based on the open-source MPICH software stack from Argonne National Laboratory. Cray MPI relies on the Generic Network Interface (GNI) and Distributed Shared Memory Application DMAPP APIs. The GNI and DMAPP APIs provide low-level communication services to user-space software stacks. These APIs allow system software stacks to fully leverage various hardware capabilities offered by the Cray Aries interconnect.

Figure 1 shows an overview of the Cray MPI software stack and also highlights specific components that have been optimized for Cray Interconnects. Cray MPI relies on the CH3/Nemesis infrastructure to implement MPI communication operations. On-node, inter-process communication is implemented within the Nemesis layer. In addition to the basic on-node communication protocols, Cray MPI also includes an Xpmem library based implementation to optimize performance of large message MPI operations. Off-node communication operations are implemented within the "GNI Netmod" layer, which directly interfaces to the low-level Cray Aries drivers and hardware components. This layer is designed to achieve high performance and extreme scalability on large Cray XC supercomputers. Cray MPICH also supports NVIDIA GPUs, a broad range of I/O optimizations and a fast and scalable job launcher via Cray PMI.

Most MPI implementations rely on Eager and Rendezvous protocols for implementation of small and large message MPI point-to-point operations. MPI point-to-point implementations in Cray MPI are based on highly tuned protocols to ensure low latency and high bandwidth for communication operations on Cray XC series systems. While the Eager threshold is typically set to 8,192 bytes, users can adjust this setting via environment variables. For messages smaller than 8,192 bytes, Cray MPI dynamically selects between two protocol implementations to optimize for memory utilization and communication performance. Similarly, for messages larger than 8,192 bytes, Cray MPI uses a combination of "Get" or "Put" based protocols depending on the payload size and the processor architecture.

Cray MPI implements a highly optimized set of collectives and one-sided Remote Memory Access (RMA) operations for Cray XC systems. RMA operations in Cray MPI are natively implemented on top of GNI transport APIs to offer very low latency, high message rates and high communication bandwidth. In addition, the RMA implementation in Cray MPI is highly optimized for multi-threaded usage scenarios. MPI one-sided communication and synchronization operations can be called concurrently from multiple user-level threads and Cray MPI offers "thread-hot" communication for the communication operations [5].

Cray MPI also offers a set of hybrid hardware and software based solutions to optimize the performance of MPI collective operations for Cray XC systems. Cray MPI leverages the Collective Engine (CE) available in the Aries hardware to optimize small message collectives (such as global reductions, barrier and broadcasts). In addition, Cray MPI also relies on a range of on-node and off-node optimizations implemented in software to improve the performance of various collectives for a broader range of message lengths [4].

For Cray XC systems based on the Intel KNL processor, Cray MPI offers a range of optimizations and features to improve the performance of parallel applications. While the KNL architecture offers support for wide vectors, the processor cores operate at slower clock frequencies and offer slower scalar processing capabilities. Since a significant portion of MPI is largely scalar, Cray MPI has been optimized to reduce the number of instructions executed in various performance critical paths. While these optimizations were specifically introduced for KNL, they are also helpful in improving the performance of Cray MPI on Intel Xeon processors. In addition, Cray MPI offers a range of environment variables to allow users to manage the MCDRAM memory offered by the KNL architecture. These features are applicable when the KNL nodes are configured in the "flat" mode. Specifically, users can request specific memory attributes (memory affinity, policy and page sizes for example) for memory regions allocated by the MPI_Alloc_mem and MPI_Win_allocate routines. In addition, users can also manage the affinity of certain memory regions allocated inside the Cray MPI library. This enables users to maximize the utilization of MCDRAM memory for application-level data by forcing MPI's internal memory to be allocated on DDR.

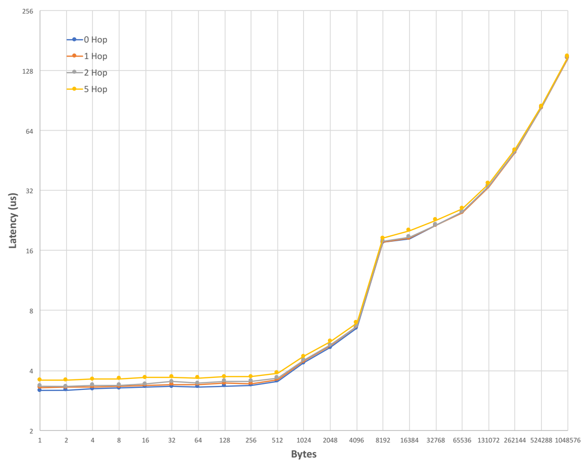

Figure 2: OSU latency benchmark results for multiple hops

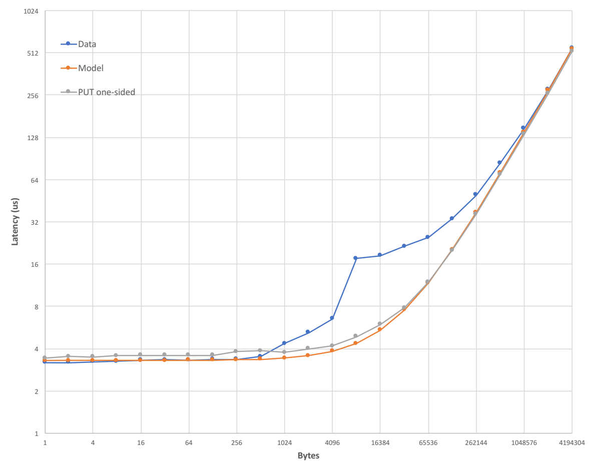

Figure 3: OSU latency benchmark data and model along with MPI_Put data

MPI POINT-TO-POINT PERFORMANCE AND MODEL

Cray MPI MPI_Send/Recv point-to-point performance was evaluated on Theta using the OSU point-to-point latency benchmark (osu_latency) which measures and reports one way latency using a ping point test. On Theta network nodes may be different distances or hops away from one another. Two nodes directly connected to the same Aries router are considered to be zero hops away from one another. Two nodes in the same group that lie on the same column or row are one hop away from each other since the message will traverse one network link. For nodes in the same group that are not on the same row or column two hops, or link traversals, are required. Finally, nodes that lie in different groups can be a maximum of five hops away from one another, since the message may be required to transit two links within the group to reach the router containing a link to the second group and then transit two more links within the destination group to reach the intended node. Each hop a message takes adds latency. Cray reports approximately 100 ns of latency per hop for Aries.

The OSU latency test was performed multiple times for nodes different hop counts away from one another in order to experimentally assess the impact of hop distance on point-to-point latency. The results are shown in Figure 2. While the overall latency is slightly higher for the five hop case it may be seen that the number of hops a messages takes does not significantly change the message latency. Message latencies on Theta are approximately 3.3 ns for small message and increases linearly for large messages. A kink in the latency curve may be noted centered around a message size of 8192 bytes.

Given the shape of the latency curve in Figure 2 it is expected to be well described by a simple Hockney [2] or "postal" model of message latency which takes the form of

$$l = \alpha + \beta n \tag{1}$$

where l is the overall message latency, α is the zero byte message latency and β is the latency per byte sent. An equation of this form was fit to the OSU latency data shown in Figure 2 and resulted in the values:

$$\alpha = 3.3$$
$$\beta = 0.0013$$

The resulting curve is shown in Figure 3. The equation may be seen to be a good fit to the experimental data for large and small messages but diverges from the experimental results in the region centered around a message size of 8192 bytes. This is the point where the two-sided MPI messaging protocol in Cray MPI switches from eager to rendezvous. For comparison the results of the OSU one sided put latency benchmark (osu_put_latency) are also plotted. MPI_Put calls do not contain a similar protocol transition and the resulting benchmark results produce a smoother curve without the kink seen in the two-sided results. The OSU put results show close agreement with equation (1) across the full range of message sizes. This provides evidence that the kink in the OSU two-sided latency data from Figure 2 is a result of the underlying eager to rendezvous protocol switch. While Equation (1) fails to capture the impact of the protocol switch it does accurately model the capabilities of the underlying network and the performance of Cray MPI two-sided point-to-point messaging in regions away from the location of the protocol switch.

The point-to-point messaging on Theta is optimized to use either the Eager or Rendezvous protocols depending on the size of the message sent.

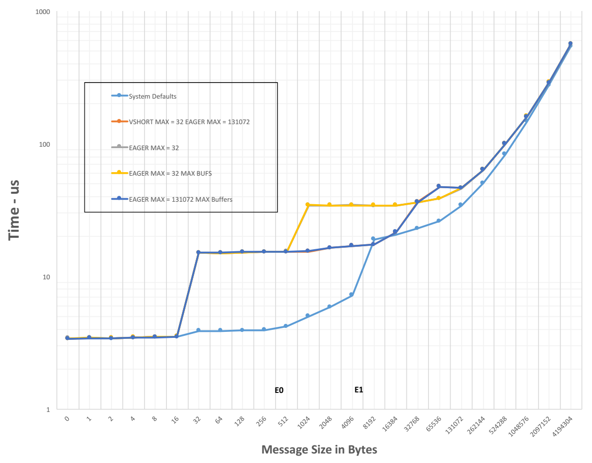

Figure 4: Effect of eager and rendezvous cutoffs on latencies for point-to-point communication

Algorithm	Range	Cost
Binomial	small messages or small rank count	$l = (\alpha + \beta n) log_2(p)$
Scatter with doubling allgather	medium messages and power of 2 ranks	$l = 2\alpha log_2(p) + 2n\beta(\frac{p-1}{p})$
Scatter with ring allgather	all other message or rank counts	$l = \alpha(log_2(p) + p - 1)$ $+2n\beta(\frac{p-1}{p})$

Table 2: MPICH MPI_Bcast algorithms and latency estimates

The Cray MPI implementation supports two Eager protocols, referred to as E0 and E1, and two Rendezvous protocols, referred to as R0 and R1. In the E0 protocol, a posted send executes immediately by depositing the message into a mailbox at the receiver's end. On Theta, E0 is used for message sizes of 0 bytes to as high as 8192 bytes. The default max value is set at 8144 bytes. The max size for E0 can be set using the $MPICH\_GNI\_MAX\_VSHORT\_MSG\_SIZE$ environment variable. The allowable range for this variable is 32 bytes to 8192 bytes, and it can be changed in 16 byte increments. Similarly, the environment variable $MPICH\_GNI\_MAX\_EAGER\_MSG\_SIZE$ controls the E1 window. The allowable range is 0 to 131072 bytes, and it can be changed in 1024 byte increments. The default is set at 8192 bytes. The R0 channel uses a GET protocol, and the size window is controlled by the environment variable $MPICH\_GNI\_GET\_MAXSIZE$. The range is 16K to 16M bytes changeable in 1024 byte increments. The default is set at 4MB. By default, all messages greater than 4MB are transferred under the R1, PUT based semantics, protocol.

The default settings for the eager and rendezvous cutoffs were used to produce the results in Figure 2. It is therefore of interest to examine how the E0, E1, R0, and R1 protocol cutoffs might impact these measured latencies. The results are depicted in Figure 4. As expected, switching from E0 to E1 for message sizes below 8192 bytes leads to significantly increased latencies. Similarly, replacing R0 in its default range with either E1 or R1 leads to increased latencies. Overall, R1 is found to have the highest latencies, while E1 is found to have latencies between the R0 and R1 numbers.

MPI COLLECTIVES PERFORMANCE DATA AND MODELS

Parallel applications make use of both MPI point-to-point and collective routines. MPI collective routines are internally implemented using a series of point-to-point message exchanges based on a variety of collective implementation patterns. Different algorithms, or patterns of point-to-point message exchange, are utilized for each MPI collective routine and often the same MPI collective will utilize one of several different algorithms based on the message size and the number of ranks in the communicator. As an example, inspection of the open source MPICH v3.2 library shows that it implements MPI_Bcast using three different algorithms depending on the message size and number of ranks. A brief description of the three algorithms is given in Table 2 along with the conditions under which a given algorithm is used. Cost or latency estimates for MPI collective implementations have been developed [7] using point-to-point message latencies modeled with the Hockney model. These cost estimates for the algorithms used for MPI_Bcast are also shown in Table 2. Cray provides a propriety MPI library that is based on MPICH but includes optimized MPI collective routines. The collectives routines that have been modified and optimized in the Cray MPI implementation are shown in the list below:

- MPI_Allreduce
- MPI_Bcast
- MPI_Barrier
- MPI_Alltoall

Routine	Relative Call Frequency
Allreduce	5000
Bcast	2500
Barrier	500
Alltoall	500
Alltoallv	250
Reduce	75
Allgatherv	25
Other collectives	<1

Table 3: Relative frequency of MPI collective routine calls by applications run on ALCF systems

- MPI_Alltoallv
- MPI_Allgather
- MPI_Allgatherv
- MPI_Gatherv
- MPI_Scatterv
- MPI_Igatherv

For single word messages the routines MPI_Allreduce, MPI_Barrier, and MPI_Bcast can make use of the hardware collectives engine in the NIC for further performance improvements when the Cray DMAPP library is linked with the application. While the Cray collective implementations are proprietary, and therefore the specifics of the algorithms cannot be examined, the overall performance of the collectives is expected to have similar characteristics to those in MPICH with $log_2(p)$ dependence on the number of processors, p, and linear dependence on the message size, n.

While MPI provides over a dozen collective routines some routines are found be used significantly more frequently in typical HPC workloads. Table 3 shows the relative frequency of collective calls made by a representative sample of applications running on the Argonne Leadership Computing Facilities Blue Gene/Q system Mira over a two year period. The frequency of the calls in the workload has been normalized relative to that of the less frequently called routines. The data shows that the most heavily utilized routines are called hundreds to thousands of times more frequently by applications in the workload than are the least commonly used routines. The performance on Theta of three of the mostly commonly used routines, MPI_Allreduce, MPI_Bcast, and MPI_Barrier is evaluated below.

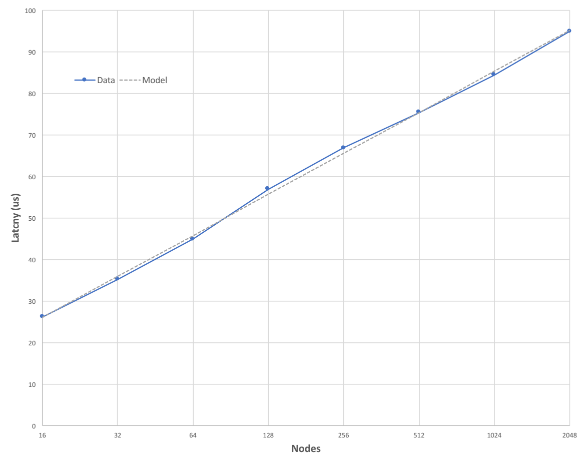

Figure 5: OSU barrier performance data and model

The results of running the OSU MPI_Barrier benchmark (osu_barrier) on Theta across a node count ranging from 16 to 2048 are shown in Figure 5. These result were obtained using one rank per node and with no other jobs running on the system. The horizontal axis is plotted using a log_2 scale and the latency of the barrier operation shows a distinct log dependence on the number of nodes. An equation of the form

$$l = \alpha + \beta \cdot log_2(p) \tag{2}$$

was fit and the values for α and β were determined to be

$$\alpha = -13.5$$
$$\beta = 9.87$$

The corresponding model line is plotted in Figure 5 and shows very good agreement with the benchmark results.

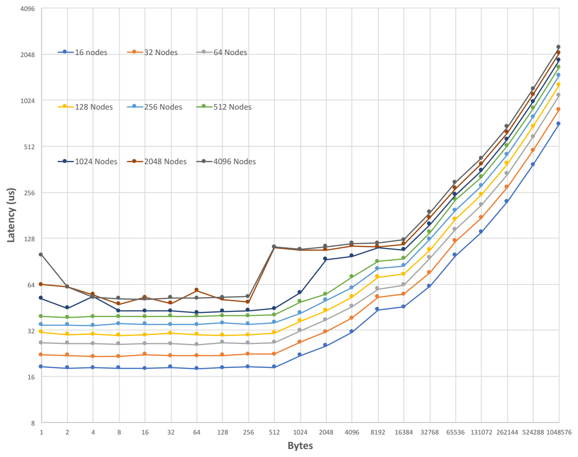

Figure 6: OSU broadcast performance data

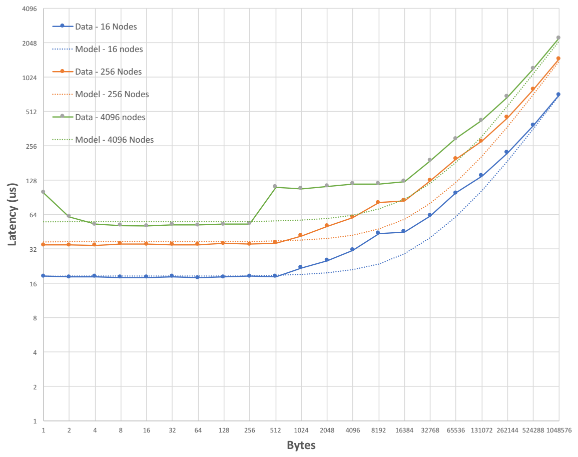

Figure 7: OSU broadcast performance data and model

The results of running the OSU MPI_Bcast benchmark (osu_bcast) on Theta are shown in Figure 6. The horizontal axis shows the number of bytes in the broadcast message and the different lines represent MPI_Bcast latencies for different numbers of nodes. The general shape of the individual latency curves is similar to that of the ping-pong latency results shown in Figure 2, including the kink at the switchover point between the eager and rendezvous protocols. Similar to MPI_Barrier the dependence on the number of nodes has a log_2 relation. An equation of the form

$$l = (\alpha + \beta \cdot n)log_2(p) \tag{3}$$

was fit and the values for α and β were determined to be

$$\alpha = 4.6$$
$$\beta = 0.0016$$

Figure 7 shows the model equation compared to the benchmark results. Only three three node counts are shown for the purposes of clarity but the results are similar across the full node range. Good agreement may be observed for lower and higher message sizes with appreciable error in the middle of the range, which is attributable to the protocol switch from eager to rendezvous. At higher node counts the protocol switch occurs at lower messages sizes which accounts for the leftward shift of the kink for the 4096 node curve.

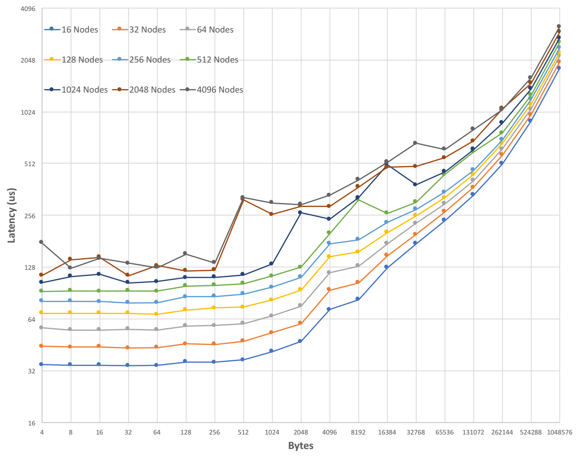

Figure 8: OSU Allreduce performance data

Finally the results of running the OSU MPI_Allreduce benchmark (osu_allreduce) on Theta are shown in Figure 8. The horizontal axis shows the number of bytes in the message and the different lines represent MPI_Allreduce latencies across different numbers of nodes. Again the general shape of the individual latency curves show a linear dependence on the number of bytes and includes the kink at the switchover between the eager and rendezvous protocols. Similar to MPI_Bcast the dependence on the number of nodes has a log_2 relation. An equation of the form

$$l = \gamma + \delta \cdot n + (\alpha + \beta \cdot n)log_2(p) \tag{4}$$

was fit and the equation parameters were determined to be

$$\gamma = -24$$
$$\delta = 0.0012$$
$$\alpha = 13.6$$
$$\beta = 0.00012$$

Figure 9 shows the model equation compared to the benchmark results. Only five node counts are shown for the purposes of clarity but the results are similar across the full node range. Good agreement may again be observed for lower and higher message sizes with appreciable error in the middle of the range attributable to the protocol switch.

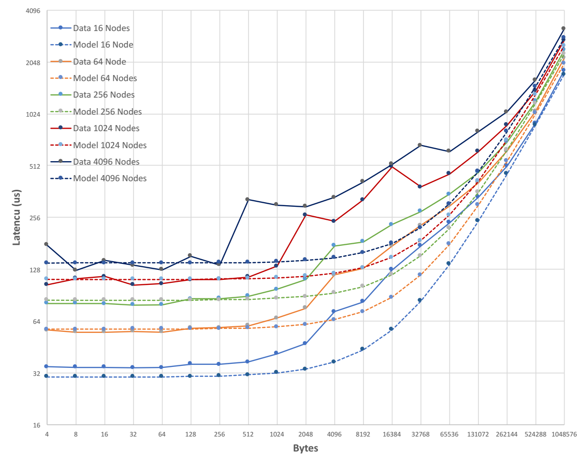

Figure 9: OSU Allreduce performance data and model

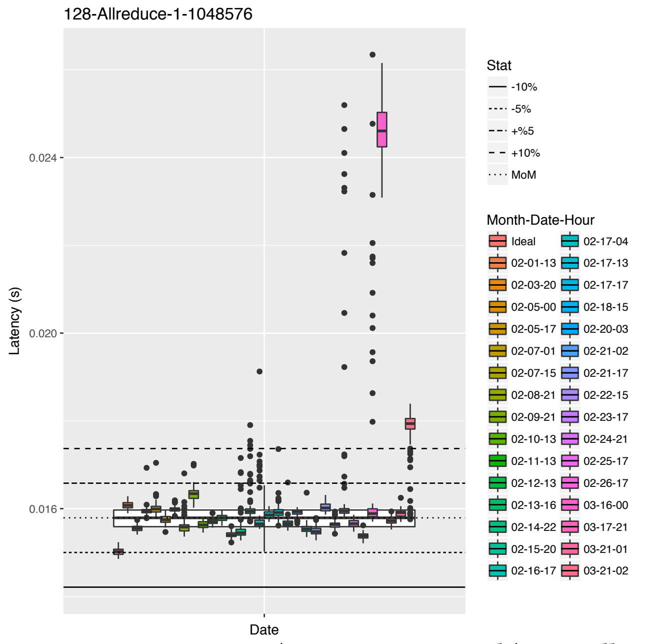

Figure 10: 128 processes (one process per node) MPI Allreduce Latency with a message of 1M DOUBLE (8MB) data

IMPACT OF NETWORK CONGESTION

The MPI point-to-point and collective results shown earlier were obtained without interference from other jobs running on the system. An earlier study [1] has shown the impact of network congestion on the latency of collective operations on Theta. In this work the OSU MPI Allreduce benchmark (osu_allreduce) was run repeatedly on different days in different jobs. For comparison the same benchmark was run in an isolated environment where no other job are running on the system. Figure 10 shows the latency for MPI_Allreduce with a message size of 1M doubles on 128 processes (using 1 process per node) on Theta. Within each job, the collective call is repeatedly run 100 times; each box in the figure represents the variability across those 100 runs. The run-to-run variability for different days is shown in the different boxes, with the horizontal axis representing different days. The bottom and top of the box are the 25th and 75th percentile respectively, and the band near the middle of the box is the 50th percentile, or the median. The middle horizontal line shows the median of medians, and the other four horizontal lines show +/-5% and +/-10% difference from the median of medians. An overall variation in latency of 35% was observed, with the lowest latency occurring when no other jobs were present. Latencies variations for smaller message sizes were found to be even higher.

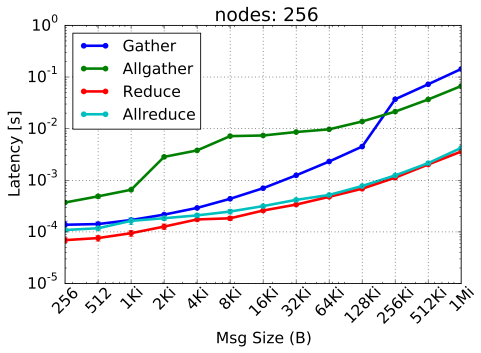

Figure 11: Scaling of Gather, Allgather, Reduce and Allreduce collectives with different message sizes for the 256 node size

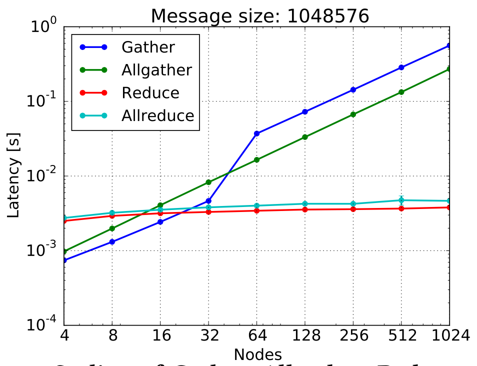

Figure 12: Scaling of Gather, Allgather, Reduce and Allreduce collectives with different node sizes for the 1MB message size

Performance consistency guidelines for MPI collectives have been established [3] and these guidelines may be used to evaluate the performance consistency of MPI collectives in both isolated and congested network environments. These performance guidelines define a performance expectation based on semantic functionality of the collectives, for example, one performance guideline states that a call to MPI_Allgather on n data elements should "not be slower" than a combination of a call to MPI_Gather with n data elements followed by a call to MPI_Broadcast with n data elements. Another guidelines says that a many-to-one collective should always perform better than a semantically related many-to-many counterpart, for example, an MPI_Gather should have less latency than an MPI_Allgather.

When run on an unloaded system no significant MPI performance consistency violations were observed for the Cray MPI on Theta. However with other jobs present regular performance violations occurred. While the precise details of the violations vary from run to run, on a loaded system the same violations appear regularly, possibly indicating that some collective

	1	2	4	8	16	32	64	128	256	512	1024	2048	4096	8192	16384	32768	65536	131072	262144	524288	1048576
MPI_Allgather ≤ Alltoall												✗	✗	✗	✗	✗	✗	✗		✗	
MPI_Allgather ≤ Allreduce						✗						✗									
MPI_Allgather ≤ Gather + Bcast	✗	✗	✗			✗	✗	✗				✗	✗	✗	✗						

Table 4: Performance guideline violations for Allgather in Cray MPI using 256 processes (1 process per node)

implementations are more susceptible to the impact of network congestion than others. As an example of the types of violations observed on a loaded system, the performance of two many-to-one collectives and their related many-to-many counterparts on 256 processes is shown in Figure 11. The benchmarks were run on 256 nodes with 1 process per node. A few performance guideline violations may be observed; MPI_Gather has more latency than MPI_Allgather for certain message sizes, and the scaling of MPI_Allgather with increasing message size is inconsistent. The performance guideline violations for the Allgather across different message sizes on 256 nodes are presented in Table 4. The performance of the Allgather is compared with the different collective combinations that essentially achieve the same functionality. Allgather has higher latency than Alltoall. This is true across all the node sizes tested for messages sizes between 4KB and 512KB. Figure 12 shows the scaling of few of the MPI collectives (Gather, Allgather, Reduce, Allreduce) for a specific message size (1MB) with increasing node size. While the scaling curve for Allgather has a consistent slope, the scaling curve for Gather is inconsistent with a spike at 64 nodes.

CONCLUSION

MPI point-to-point latency on the Cray XC40 system Theta may be well represented by a simple Hockney "postal" model. The minimum message latency for one or two sided point-to-point messages was found to be approximately 3.3 ns and the latency was linearly dependent on the message size for larger messages. The model represents the MPI latency accurately for most message sizes, except in the region close to the switch from an eager to a rendezvous protocol which occurs near 8K bytes. The Hockney models was found to be highly accurate for MPI_Put latencies which do not undergo a protocol switch. The protocol transition points may be altered through environment variable settings, but no improvements in latency were found for different settings. The Cray XC Aries Dragonfly topology has a distance between nodes of zero to five network hops, however the impact of the number of hops on the measured latency was found to be minimal.

MPI libraries implement collective calls using a variety of underlying point-to-point message patterns. Different collectives utilize different patterns and individual collective calls may utilize several different patterns for different rank and byte counts. The collective implementations in the open source MPICH library were examined and found to have well established latency models which generally exhibit a log_2 dependence on the number of ranks and a linear dependence on message size. Cray MPI provides optimized collectives implementations for a number of collective routines. The latency models for these routines could not be a priori determined due to the proprietary nature of the implementation. While MPI defines over a dozen distinct collective calls, analysis of a representative workload shows that MPI_Allreduce, MPI_Bcast, MPI_Barrier, and MPI_Alltoall are the most frequently used, and are called hundreds to thousands of times more often than other collective routines. The OSU MPI benchmarks were used to determine the latency of MPI_Allreduce, MPI_Bcast, and MPI_Barrier on Theta and models were fitted that accurately represent the latencies of these routines for message sizes away from the eager to rendezvous transition point. Much like the MPICH collectives, the latencies of the Cray MPI collectives that were studied were found to have a log_2 dependence on the number of ranks and a linear dependence on the messages size.

The results of previous studies on Theta that evaluated the impact of network congestion on collectives performance were discussed, with relevant results highlighted showing that collective latencies can vary by 35% or more due to congestion. Additionally, earlier evaluations of the relative performance consistency of MPI collectives were updated with results from runs performed without network contention from other jobs. These results showed that most of the performance inconsistencies previously observed were not present without network contention. However inconsistencies have been found to persistently occur when network contention is present, indicating that some collective algorithm implementation patterns may be more susceptible to the impact of congestion than others.

REFERENCES

[1] Sudheer Chunduri, Kevin Harms, Scott Parker, Vitali Morozov, Samuel Oshin, Naveen Cherukuri, and Kalyan Kumaran. 2017. Run-to-run Variability on Xeon Phi Based Cray XC Systems. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis (SC '17)*. ACM, New York, NY, USA, Article 52, 13 pages. https://doi.org/10.1145/3126908.3126926

[2] R. Hockney and C. Jesshope. 1981. *Parallel Computers: Architecture, Programming and Algorithms*.

[3] Sascha Hunold, Alexandra Carpen-Amarie, Felix Donatus Lübbe, and Jesper Larsson Träff. 2016. Automatic Verification of Self-consistent MPI Performance Guidelines.. In *Euro-Par*. 433–446.

[4] Krishna Chaitanya Kandalla, David Knaak, and Katie L. McMahon. 2015. Optimizing Cray MPI and Cray SHMEM for Current and Next Generation Cray-XC Supercomputers. In *Proceedings of the 2015 Cray User Group Meeting*.

[5] Krishna Chaitanya Kandalla, Peter Mendygral, Nicholas J. Radcliffe, Bob Cernohous, David Knaak, and Katie L. McMahon. 2016. Optimizing Cray MPI and SHMEM Software Stacks for Cray-XC Supercomputers based on Intel KNL Processors. In *Proceedings of the 2016 Cray User Group Meeting*.

[6] Sébastien Rumley, Meisam Bahadori, Robert Polster, Simon D Hammond, David M Calhoun, Ke Wen, Arun Rodrigues, and Keren Bergman. 2017. Optical interconnects for extreme scale computing systems. *Parallel Comput.* 64 (2017), 65–80. https://doi.org/10.1016/j.parco.2017.02.001

[7] Rajeev Thakur, Rolf Rabenseifner, and William Gropp. 2005. Optimization of Collective Communication Operations in MPICH. *Int. J. High Perform. Comput. Appl.* 19, 1 (Feb. 2005), 49–66. https://doi.org/10.1177/1094342005051521

[8] Jesper Larsson Träff. 2012. mpicroscope: Towards an MPI benchmark tool for performance guideline verification. In *European MPI Users' Group Meeting*. Springer, 100–109.

[9] Jesper Larsson Träff, William D Gropp, and Rajeev Thakur. 2010. Self-consistent MPI performance guidelines. *IEEE Transactions on Parallel and Distributed Systems* 21, 5 (2010), 698–709.